Firewall Applications & Anomaly Detection for Network Security

Glenn Skawski
School of Computing, Informatics, and Decision Systems Engineering
Arizona State University
Tempe, Arizona, Maricopa County
gskawski@asu.edu

Introduction

Herein are descriptions and results of four network security applications that are possible solutions to two common goals of any network architecture, specifically, implement security measures that: 1) dictate how devices on a private network communicate and access data (*iptables* and OpenFlow-based firewalls); and 2) detect and mitigate network attacks originating from an external malicious agent (port security and anomaly detection firewalls). Additionally, are descriptions of different network setups to which the security measures were applied.

First exercise is using Linux firewall iptables which is a packet filtering method which is a fundamental network security concept having numerous applications within any given network architecture [1]. Next is implementation of a firewall on a software defined network which has the benefit of simplifying network complexity by providing network security at a virtual switch for Layer 2 and 3 of the TCP/IP Model. The firewall here includes static flow table entries directing traffic on a private network [2]. Building onto SDN security is creating a firewall that can also dynamically allocate flow entries in a layer 3 switch in real-time which in this case is demonstrated by blocking a Denial-of-Service attack [3]. Finally, using a feed-forward neural network to create an anomaly detection model for network traffic that can detect a swath of different attacks and shows how security can be automated [4].

1 Packet Filter Firewall in Virtual Network

The packet filter firewall was applied to a virtual network composed of a client and gateway/server virtual machine (VM) using iptables inside the gateway/server VM. iptables' rules were created to satisfy a set of security requirements. The gateway/server is configured to act as the gateway router, client's firewall for packet filtering and forwarding, and host a local HTTP web service. The virtual machines were hosted locally using VirtualBox and Ubuntu operating system.

1.1 Virtual Network Setup

Overview of the virtual network setup is the client and gateway/ server be connected on the private network 10.0.2.0 on interface *enp0s3* with IP addresses 10.0.2.5 and 10.0.2.6, respectively. The gateway/server is connected to the internet existing on network 10.0.1.0 on interface *enp0s8* and performs specific routing and packet filtering for the client. General steps taken to create the network is first set the client's default gateway to the gateway VM and change the client's private network to static. This can be accomplished by modifying the file */etc/netplan/01-network-manager-all.yaml*. Next configure in the gateway/server VM routing table so that it's default gateway of 10.0.1.1 has the highest priority (e.g. lowest interface metric in routing table) via *ifmetric* command. Also enable package forwarding in gateway VM by modifying the file */etc/sysctl.conf* file.

The webservice is created locally on the gateway/server VM by first installing *apache2*. Configuration and variable files are provided in the installation that can be used for hosting a simple webservice. In this case it is specified to have the local web service contain "Welcome" in its title accomplished by modifying the file */var/www/html/index.html* [5].

1.2 iptables Firewall

The packet filter firewall is created in the gateway/server VM using Linux firewall iptables to implement a set of security policies, specifically: client can access local web service and ping 8.8.8.8; client can't ping gateway/server; gateway can access local web service; and drop all remaining network access. The first step is enacting a whitelist which will prevent all access to the local network unless specified (e.g. setting INPUT, OUTPUT, and FORWARD to DROP). To allow client access to 8.8.8.8 add a POSTROUTING entry to MASQUERADE client when making ping request to 8.8.8.8 and add FORWARD entries that accept icmp packets to/from 8.8.8.8. To allow client access to the local webservice add INPUT entry to accept TCP requests from client destined to gateway/server's IP and port 80 and add OUTPUT entry accepting gateway/server response to client.

2 OpenFlow-Based Stateless Firewall in a SDN

An OpenFlow-based stateless firewall was applied to a software defined network (SDN) composed of a switch, two controllers, and four host containers which were all hosted locally in a virtual machine using VirtualBox and Ubuntu operating system. The switch is responsible for the hosts' Layer 2 and Layer 3 switching / routing and communicating with controllers for packet filtering and forwarding directions via flow table rules obtained using OpenFlow protocol [6].

2.1 SDN Setup

SDN setup includes first initiating the network by executing in sequence the programs Open vSwitch (OVS), a POX controller listening on port 6655 and configured for L2 switching and L3 routing via *l2_learning* and *L3Firewall.py* files, respectively, in the POX forwarding folder, a POX controller listening on port 6633, and a Mininet / Containernet topology to the SDN specifications. The SDN topology can be initiated one of two ways: 1) Using Mininet commands in the command line terminal to create four containers, integrate the two remote controllers, and use the OVS; or 2) using a custom topology written in python that when executed integrates all the devices and allows user to include addition network specifications such as IP addresses, MAC addresses, etc [7]. Changing network specifications using the former topology method requires using the xterm command in Containernet to individually access containers and make changes.

2.2 Stateless Firewall Design

The stateless firewall functionality resides in the L3Firewall.py application written in python. The firewall application contains the L2 and L3 flow table rules / entries (from l2firewall.config and l3firewall.config files) and methods that control OpenFlow events between the switch and controller. The firewall application's methods / events include *_handle_ConnectionUp* upon establishment of new control channel [lab3 5], *_handle_PacketIn* for a switch-controller communication event when the switch requests direction on a packet not matching any current flow entries, and *installFlow* for adding a new flow table entry in the switch [6]. The flow table rules are static, written in advance of SDN initiation, located in the config files, and based on a specified security policy for the SDN. Instantiation of the SDN is when the POX controller reads the config files containing the flow entries, processes rules into messages, and the switch adds rules to its flow tables based on controller messages. At layer 2, the switch will be provided filtering entries to drop packets with specific MAC destinations and sources, such as in Figure 1, dropping packets going to ::04 from ::02.

```
id,mac_0,mac_1
1,00:00:00:00:00:02, 00:00:00:00:00:04
```

Figure 1: Flow rule example in l2firewall.config file

At layer 3, the switch can be provided filtering and forwarding entries based on packet header information that includes MAC / IP destination and source, source and destination port, and protocol. Also specified is the rule priority for the switch to sequence through for possible matches when receiving a packet. In this case, the firewall application will drop incoming packets that match flow table rule entries, such as in Figure 2, dropping pings between host #1 and #3, dropping HTTP traffic from host #2, etc.

```
priority,src_mac,dst_mac,src_ip,dst_ip,src_port,dst_port,nw_proto
1,any,any,192.168.2.10,192.168.2.30,1,1,icmp
3,any,any,192.168.2.20,any,80,any,tcp
```

Figure 2: Flow rule examples in l3firewall.config file

3 Block DOS Attack on SDN Controller

A denial of service (DoS) attack on an SDN switch is simulated with an objective to create a firewall application capable of detecting and blocking the attack in real time.

3.1 SDN and DoS Attack Setup

SDN device specifications and setup methodology is similar to that detailed in Section 2.1 SDN Setup with exception to using one POX controller and not including the *l2firewall* and *l3firewall* config files. The attack is simulated in the SDN by host #1 executing the command *hping3 <host #2 IP> -c 10000 -S --flood --rand-source -V* [3]. This command will flood the switch with TCP packets appearing to originate from numerous randomly generated IP addresses and without a firewall to block the attack will prevent all communication needing to go through the switch.

3.2 Firewall with Port Security

The firewall is designed to block a DoS attack by implementing a port security strategy. Port security is a layer 2 application monitoring and controlling ingress port access based on packet's MAC source. Port security is coded into the *L3Firewall.py* application and functions by using OpenFlow events to detect the DoS attack and create messages of new flow table entries for the switch to add which can then block the attack [8]. The difference between the port security firewall and the firewall described in Section 2.2 is here flow table entries are created by the controller in real-time based on application logic instead of statically in advance of network startup.

The port security logic is written in the firewall's (*L3Firewall.py*) *_handle_PacketIn* method as this will be called every time the switch receives a packet not matching its flow entries. This logic first creates a port table when starting the POX controller which is an array of unique source MAC address, source IP address tuples. Unique MAC / IP pairs are added to the port table when a device with a MAC address not in the port table uses the switch for the first time. Subsequent attempts by the same device will be allowed as long as the packets MAC / IP source address match what is in the port table. If that same device attempts to spoof the switch with different source IP addresses, the firewall will recognize the devices MAC & IP do not match and will send a message to the switch directing it to add a new flow entry that drops packets from that device for a given duration.

4 Feed-Forward Neural Network (FNN) for Network Traffic Anomaly Detection

A Feed-forward Neural Network (FNN) was used to create a network anomaly detection model capable of classifying traffic as normal or as an attack (DoS, Probe, U2R, or R2L). The model is trained and tested using the NSL-KDD dataset of modern-day internet traffic [9].

4.1 FNN Training and Testing

The first general step for creating a classification model is pre-processing the data (*KDDTrain+.txt* and *KDDTest+.txt* from NSL-KDD). The first pre-processing step is feature mapping using *categoryMapper.py* on the entire data set to map categorical features to unique integer values. Next is create scenarios using *dataExtractor.py* of customized training and testing datasets with respect to attack classes. Each scenario's training and testing data will be used as inputs in the FNN application(*fnn_sample.py*) which conducts additional pre-processing followed by FNN training and testing using *keras*. Pre-processing in the FNN application is done by *data_preprocessor.py* and includes feature mapping features, scaling, *One-Hot Encoding*, mapping ground truth to return the data structures *X_train, y_train, X_test, y_test* that are used as inputs by the FNN [10]. The FNN outputs accuracy and loss metrics during model training and a confusion matrix summarizing results from the trained model's traffic classification predictions of the test data and how predictions compare to the test data's ground truth [11, 12].

4.2 Anomaly Detection Model Accuracy

The purpose of creating running FNN on different scenarios was to evaluate how a model's anomaly detection accuracy might vary with respect to the composition (e.g. attack types) of training and testing datasets used by the FNN. Scenarios were designed to test two general methods for training and testing the FNN: 1) training and testing dataset have similar attack types; and 2) datasets are dissimilar in that the model is tested with attacks that it was not trained on. The average accuracy from scenarios designed using the of the former methodology is approximately 89% while the average accuracy from scenarios designed using the latter methodology is approximately 76%. The accuracy results indicate model performance does depend on training and testing data content.

5 Lessons Learned

The packet filtering firewall exercise was my first glimpse into not only networking security, but networking in general. First gained insight from the exercise was applying networking testing and debugging tools that allowed for better understanding of the tools' usefulness and applications. Notable tool is the routing table for its information content and methods to configure it. The packet filter firewall depended on the network being appropriately configured which was heavily dependent on a correct routing table. Practicing with iptables gave feedback on its command structure and available inputs to produce needed functionality. I continually reviewed and re-evaluated my understanding of packet processing in order to correctly apply rules when using iptables.

Setting up the SDN had the challenge of getting accustomed with using and configuring the different software components in order to build the virtual network. It is obvious the benefits of designing a SDN in a virtual environment with virtual devices, but one must first deal with an additional layer of complexity that is understanding and merging networking concepts with their corresponding virtual components. I tried Mininet and practices similar to the lab as preparation for this class, but was unsuccessful due to not understanding meanings, abstractions, etc. of involved tools. Through experience gained with the lab, I can use the learned tools to focus on network design and security. With the addition of this lab, I am getting a sense for big picture concepts such as containers vs virtual machines, private and public networks, layer 3 routing, layer 2 data linking, etc. I can begin to reflect on past concepts, see how they look applied, and eventually be proficient and adaptable in deploying networking tasks. At first glance this project was more intimidating due to the degree of virtualization and new tools, but the exercises sufficiently and quickly caught me up.

The DoS simulation exercise provided various avenues of feedback for visualizing and utilizing OpenFlow protocol to manage network traffic at an SDN switch that is communicating with a controller. Real-time communication between the switch and controller can be viewed through either the controller (e.g. terminal where the POX controller python program is running) or the switch via the dump-flows command. Utilizing OpenFlow protocol is done at the lower levels of the SDN architecture, specifically by programming function-specific sub-components located within the POX controller's forwarding component. This included establishing a new switch-controller connection, switch packet-in to controller, OpenFlow messages, flow table modifications, packet matching, and OpenFlow actions [6]. Combining OpenFlow protocol and a medium in which it can be applied sets the stage for realization of infinite possibilities to create a secure SDN. This lab touched on securing it from a DoS attack which would be one of many requirements needing to be implemented to ensure network security. Additional reflection of gained skills beyond the lab's learning goals includes using shell scripts and preconfigured Mininet topology files to make it significantly easier to test and adjust the firewall.

The performance of generated scenario-specific FNN models were deduced by incorporating both the: 1) resulting quantitative accuracy measurements; and 2) qualitative effect that different training set scenarios had on correct test set predictions. Given that model performance depends on scenario construction, the performance analysis indicates the model is effective and adaptable when the appropriate data and methods are applied. A major subject takeaway from the lab is the importance of using good data to train the model. My previous work with machine learning models applied less robust data and could partially explain the poor results. Additionally, is the apparent benefit of the pre-processing methods used on the raw data. The benefit of one-hot encoding to avoid bias in the model is a notable beneficial pre-processing step [13]. Emphasis on good data and appropriate pre-processing allows the FNN to function and provide previously unknown knowledge with little human intervention on model parameters. In previous machine learning attempts, I realize that my focus on improving model performance through machine learning parameter optimization likely has less impact as compared to focusing on what is going into the model. Final note is the larger picture of tying machine learning to an SDN via a controller which can dynamically automate network security in real time.

6 References

[1] Huang, Dijiang. “CS-CNS-00001 – Packet Filter Firewall (Iptables).” Arizona State University, 2021.

[2] Huang, Dijiang. “CS-CNS-00101 – OpenFlow Based Stateless Firewall.” Arizona State University, 2021.

[3] Huang, Dijiang. “CS-CNS-00103 – DOS Attack on SDN Controller.” Arizona State University, 2021.

[4] Huang, Dijiang. “CS-ML-00301 – Use Feed-Forward Neural Network (FNN) for Network Traffic Anomaly Detection.” Arizona State University, 2021.

[5] Huang, Dijiang. “CS-SYS-00003 – Basic Web Service (Apache) Setup on Linux.” Arizona State University, 2021.

[6] McCauley et al. “Installing POX¶.” Installing POX - POX Manual Current Documentation, Github, 2015, noxrepo.github.io/pox-doc/html/#openflow-in-pox.

[7] Dijiang, Dijiang. “CS-NET-00009 – Containernet Lab.” CS-CNS. www.coursera.org/learn/cse548-advanced-computer-network-security/supplement/W7shJ/course-project-2-sdn-based-stateless-firewall.

[8] Mahler, David. “Introduction to OpenFlow.” YouTube, YouTube, 7 Oct. 2013, www.youtube.com/watch?v=l25Ukkmk6Sk&t=199s.

[9] Huang, Dijang. “CS-ML-00101 – Understanding NSL-KDD Dataset.” Arizona State University, 2021.

[10] Huang, Dijiang. “CS-ML-00200 – Python Machine Learning Data Processing Modules.” Arizona State University, 2021.

[11] “Loss Functions¶.” *Loss Functions - ML Glossary Documentation*, Sphinx, 2017, ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html.

[12] Team, Keras. “Keras Documentation: Accuracy Metrics.” *Keras*, Keras, 2021, keras.io/api/metrics/accuracy_metrics/.

[13] Yadav, Dinesh. “Categorical Encoding Using Label-Encoding and One-Hot-Encoder.” *Medium*, Towards Data Science, 9 Dec. 2019, towardsdatascience.com/categorical-encoding-using-label-encoding-and-one-hot-encoder-911ef77fb5bd.